8- лекция. Метод случайного леса (RandomForest). Дерево решений и бэгинг и сравнение

1. Бэггинг
2. Ансамбли
3. Бутстрэп
4. Бэггинг
5. Out-of-bag ошибка
6. Случайный лес
7. Алгоритм
8. Сравнение с деревом решений и бэггингом
9. Плюсы и минусы случайного леса

1. Бэггинг

Из прошлых лекций вы уже узнали про разные алгоритмы классификации, а также научились правильно валидироваться и оценивать качество модели. Но что делать, если вы уже нашли лучшую модель и повысить точность модели больше не можете? В таком случае нужно применить более продвинутые техники машинного обучения, которые можно объединить словом «ансамбли». Ансамбль — это некая совокупность, части которой образуют единое целое. Из повседневной жизни вы знаете музыкальные ансамбли, где объединены несколько музыкальных инструментов, архитектурные ансамбли с разными зданиями и т.д.

Ансамбли

Хорошим примером ансамблей считается теорема Кондорсе «о жюри присяжных» (1784). Если каждый член жюри присяжных имеет независимое мнение, и если вероятность правильного решения члена жюри больше 0.5, то тогда вероятность правильного решения присяжных в целом возрастает с увеличением количества членов жюри и стремится к единице. Если же вероятность быть правым у каждого из членов жюри меньше 0.5, то вероятность принятия правильного решения присяжными в целом монотонно уменьшается и стремится к нулю с увеличением количества присяжных.

Бутстрэп

Bagging (от Bootstrap aggregation) — это один из первых и самых простых видов ансамблей. Он был придуман Ле́о Бре́йманом в 1994 году. Бэггинг основан на статистическом методе бутстрэпа, который позволяет оценивать многие статистики сложных распределений.

Метод бутстрэпа заключается в следующем. Пусть имеется выборка X размера N. Равномерно возьмем из выборки N объектов с возвращением. Это означает, что мы будем N раз выбирать произвольный объект выборки (считаем, что каждый объект «достается» с одинаковой вероятностью 1N), причем каждый раз мы выбираем из всех исходных N объектов. Можно представить себе мешок, из которого достают шарики: выбранный на каком-то шаге шарик возвращается обратно в мешок, и следующий выбор опять делается равновероятно из того же числа шариков. Отметим, что из-за возвращения среди них окажутся повторы. Обозначим новую выборку через X1. Повторяя процедуру M раз,

сгенерируем M подвыборок X1,…,XM. Теперь мы имеем достаточно большое число выборок и можем оценивать различные статистики исходного распределения.
2. Случайный лес

Лео Брейман нашел применение бутстрэпу не только в статистике, но и в машинном обучении. Он вместе с Адель Катлер усовершенстовал алгоритм случайного леса, предложенный Хо, добавив к первоначальному варианту построение некоррелируемых деревьев на основе CART, в сочетании с методом случайных подпространств и бэггинга.

Решающие деревья являются хорошим семейством базовых классификаторов для бэггинга, поскольку они достаточно сложны и могут достигать нулевой ошибки на любой выборке. Метод случайных подпространств позволяет снизить коррелированность между деревьями и избежать переобучения. Базовые алгоритмы обучаются на различных подмножествах признакового описания, которые также выделяются случайным образом.
Ансамбль моделей, использующих метод случайного подпространства, можно построить, используя следующий алгоритм:

Пусть количество объектов для обучения равно N
, а количество признаков D
.
Выберите L как число отдельных моделей в ансамбле. Для каждой отдельной модели l выберите

l(dl<D) как число признаков для l. Обычно для всех моделей используется только одно значение dl
.
Для каждой отдельной модели l создайте обучающую выборку, выбрав

Dl признаков из D, и обучите модель. Теперь, чтобы применить модель ансамбля к новому объекту, объедините результаты отдельных

L
моделей мажоритарным голосованием или путем комбинирования апостериорных вероятностей.

Алгоритм

Алгоритм построения случайного леса, состоящего из

N деревьев, выглядит следующим образом:

Для каждого n=1,…,N:
Сгенерировать выборку

Xn с помощью бутстрэпа;
Построить решающее дерево

Bn по выборке Xn
:

— по заданному критерию мы выбираем лучший признак, делаем разбиение в дереве по нему и так до исчерпания выборки — дерево строится, пока в каждом листе не более

Nmin объектов или пока не достигнем определенной высоты дерева
— при каждом разбиении сначала выбирается m случайных признаков из

N исходных, и оптимальное разделение выборки ищется только среди них.

Итоговый классификатор
a(x)=1N∑i=1Nbi(x), простыми словами — для задачи кассификации мы выбираем решение голосованием по большинству, а в задаче регрессии — средним.

Рекомендуется в задачах классификации брать m=n, а в задачах регрессии — m=n3, где n— число признаков. Также рекомендуется в задачах классификации строить каждое дерево до тех пор, пока в каждом листе не окажется по одному объекту, а в задачах регрессии — пока в каждом листе не окажется по пять объектов.

Таким образом, случайный лес — это бэггинг над решающими деревьями, при обучении которых для каждого разбиения признаки выбираются из некоторого случайного подмножества признаков.
Метод случайного леса реализован в библиотеке машинного обучения scikit-learn двумя классами RandomForestClassifier и RandomForestRegressor.
class sklearn.ensemble.RandomForestRegressor(
n_estimators — число деревьев в "лесу" (по дефолту – 10)
criterion — функция, которая измеряет качество разбиения ветки дерева (по дефолту — "mse" , так же можно выбрать "mae")
max_features — число признаков, по которым ищется разбиение. Вы можете указать конкретное число или процент признаков, либо выбрать из доступных значений: "auto" (все признаки), "sqrt", "log2". По дефолту стоит "auto".
max_depth — максимальная глубина дерева (по дефолту глубина не ограничена)
min_samples_split — минимальное количество объектов, необходимое для разделения внутреннего узла. Можно задать числом или процентом от общего числа объектов (по дефолту — 2)
min_samples_leaf — минимальное число объектов в листе. Можно задать числом или процентом от общего числа объектов (по дефолту — 1)
min_weight_fraction_leaf — минимальная взвешенная доля от общей суммы весов (всех входных объектов) должна быть в листе (по дефолту имеют одинаковый вес)
max_leaf_nodes — максимальное количество листьев (по дефолту нет ограничения)
min_impurity_split — порог для остановки наращивания дерева (по дефолту 1e-7)
bootstrap — применять ли бустрэп для построения дерева (по дефолту True)
oob_score — использовать ли out-of-bag объекты для оценки R^2 (по дефолту False)
n_jobs — количество ядер для построения модели и предсказаний (по дефолту 1, если поставить -1, то будут использоваться все ядра)
random_state — начальное значение для генерации случайных чисел (по дефолту его нет, если хотите воспроизводимые результаты, то нужно указать любое число типа int
verbose — вывод логов по построению деревьев (по дефолту 0)
warm_start — использует уже натренированую модель и добавляет деревьев в ансамбль (по дефолту False)
)

Для задачи классификации все почти то же самое, мы приведем только те параметры, которыми RandomForestClassifier отличается от RandomForestRegressor

class sklearn.ensemble.RandomForestClassifier(
criterion — поскольку у нас теперь задача классификации, то по дефолту выбран критерий "gini" (можно выбрать "entropy")
class_weight — вес каждого класса (по дефолту все веса равны 1, но можно передать словарь с весами, либо явно указать "balanced", тогда веса классов будут равны их исходным частям в генеральной совокупности; также можно указать "balanced_subsample", тогда веса на каждой подвыборке будут меняться в зависимости от распределения классов на этой подвыборке.
)
class sklearn.ensemble.RandomForestClassifier(criterion — поскольку у нас теперь задача классификации, то по дефолту выбран критерий "gini" (можно выбрать "entropy") class_weight — вес каждого класса (по дефолту все веса равны 1, но можно передать словарь с весами, либо явно указать "balanced", тогда веса классов будут равны их исходным частям в генеральной совокупности; также можно указать "balanced_subsample", тогда веса на каждой подвыборке будут меняться в зависимости от распределения классов на этой подвыборке.)
Плюсы и минусы случайного леса

Плюсы:
— имеет высокую точность предсказания, на большинстве задач будет лучше линейных алгоритмов; точность сравнима с точностью бустинга
— практически не чувствителен к выбросам в данных из-за случайного сэмлирования
— не чувствителен к масштабированию (и вообще к любым монотонным преобразованиям) значений признаков, связано с выбором случайных подпространств
— не требует тщательной настройки параметров, хорошо работает «из коробки». С помощью «тюнинга» параметров можно достичь прироста от 0.5 до 3% точности в зависимости от задачи и данных
— способен эффективно обрабатывать данные с большим числом признаков и классов
— одинаково хорошо обрабатывет как непрерывные, так и дискретные признаки
— редко переобучается, на практике добавление деревьев почти всегда только улучшает композицию, но на валидации, после достижения определенного количества деревьев, кривая обучения выходит на асимптоту
— для случайного леса существуют методы оценивания значимости отдельных признаков в модели
— хорошо работает с пропущенными данными; сохраняет хорошую точность, если большая часть данных пропущенна
— предполагает возможность сбалансировать вес каждого класса на всей выборке, либо на подвыборке каждого дерева
— вычисляет близость между парами объектов, которые могут использоваться при кластеризации, обнаружении выбросов или (путем масштабирования) дают интересные представления данных
— возможности, описанные выше, могут быть расширены до неразмеченных данных, что приводит к возможности делать кластеризацию и визуализацию данных, обнаруживать выбросы
— высокая параллелизуемость и масштабируемость.

Минусы:
— в отличие от одного дерева, результаты случайного леса сложнее интерпретировать
— нет формальных выводов (p-values), доступных для оценки важности переменных

— алгоритм работает хуже многих линейных методов, когда в выборке очень много разреженных признаков (тексты, Bag of words)
— случайный лес не умеет экстраполировать, в отличие от той же линейной регрессии (но это можно считать и плюсом, так как не будет экстремальных значений в случае попадания выброса)
— алгоритм склонен к переобучению на некоторых задачах, особенно на зашумленных данных
— для данных, включающих категориальные переменные с различным количеством уровней, случайные леса предвзяты в пользу признаков с большим количеством уровней: когда у признака много уровней, дерево будет сильнее подстраиваться именно под эти признаки, так как на них можно получить более высокое значение оптимизируемого функционала (типа прироста информации)
— если данные содержат группы коррелированных признаков, имеющих схожую значимость для меток, то предпочтение отдается небольшим группам перед большими
— больший размер получающихся моделей. Требуется O(NK) памяти для хранения модели, где K — число деревьев.